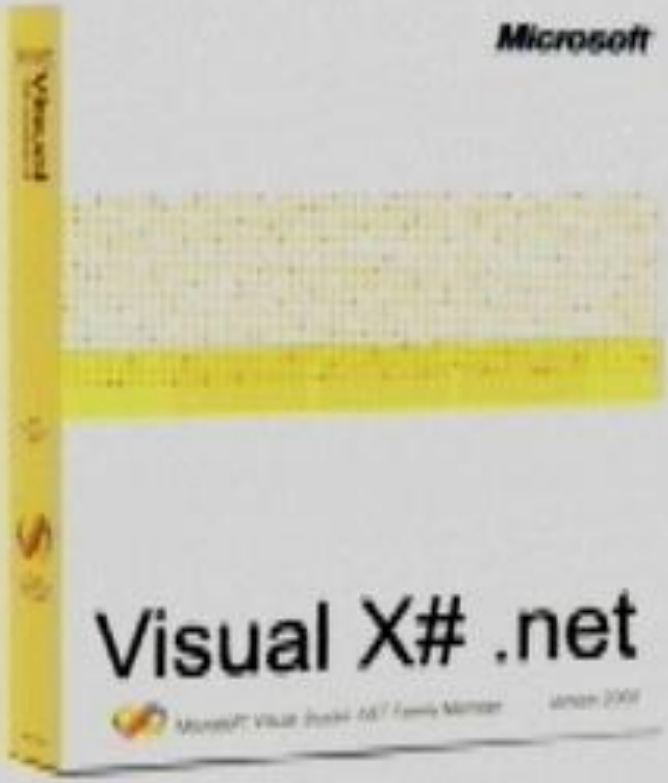
Common Compiler Infrastructure

Herman Venter
Microsoft Corporation

Among other things, it is yet another compiler compiler

X# Syntax and Semantics

Common Compiler Infrastructure

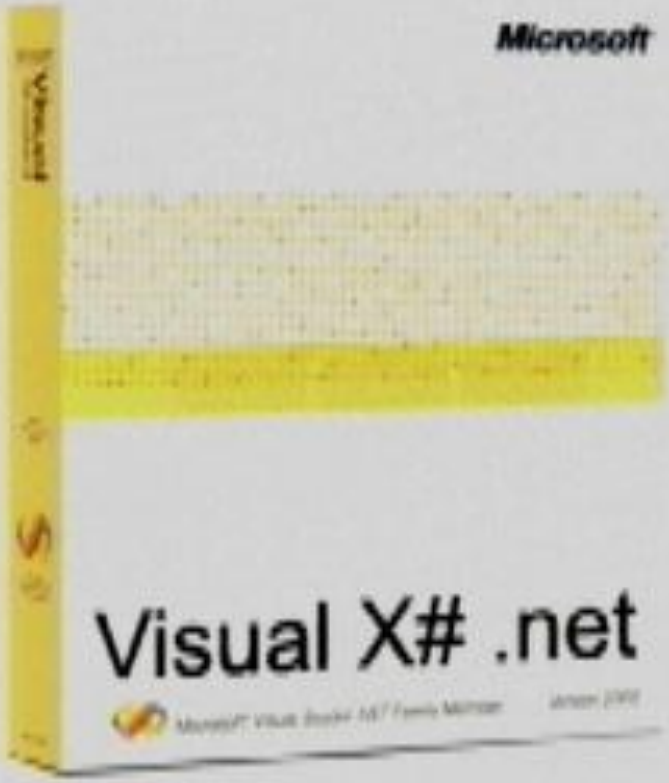


Not quite a semester project for undergrads

Among other things, it is yet another compiler compiler

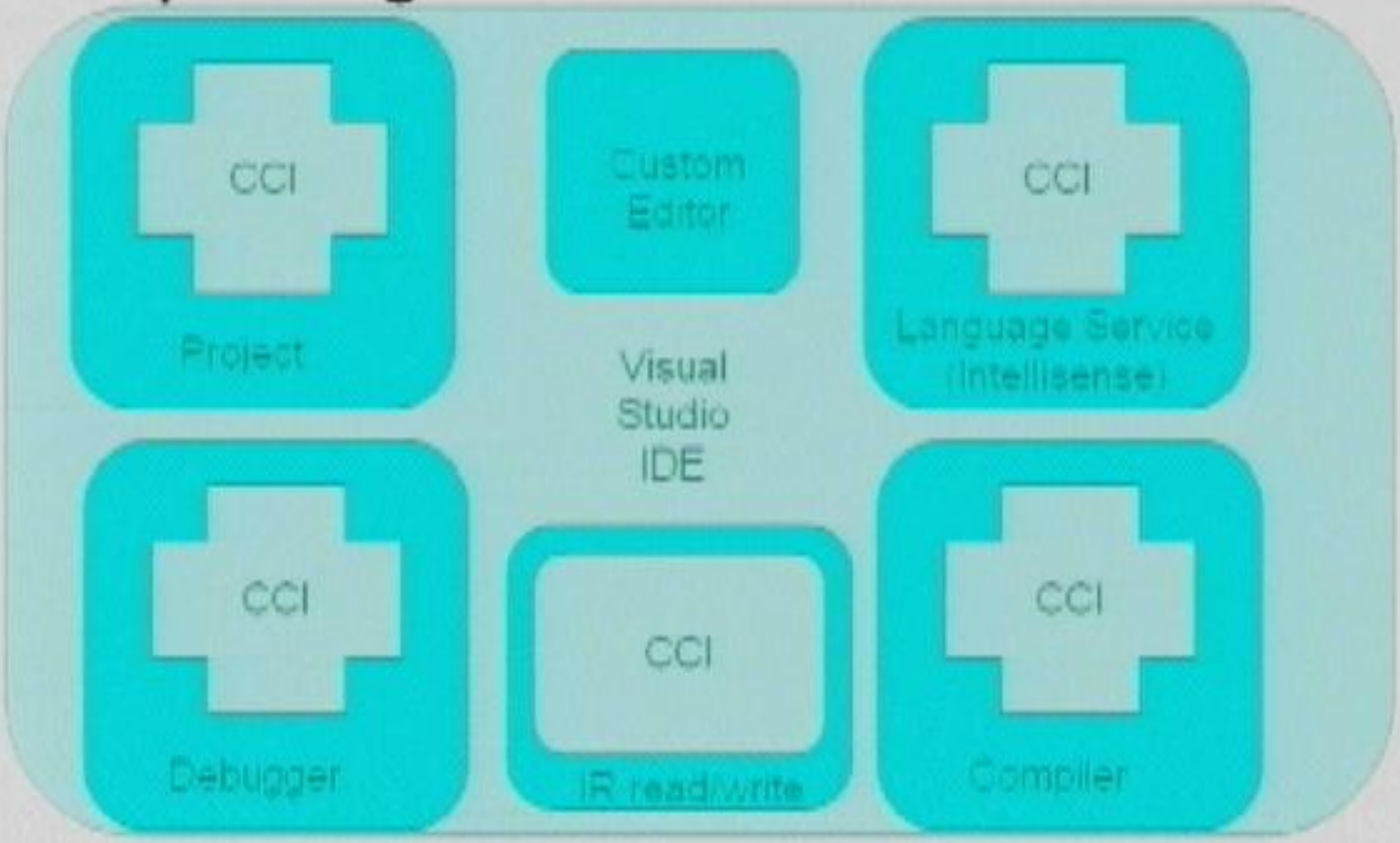
X# Syntax and Semantics

Common Compiler Infrastructure

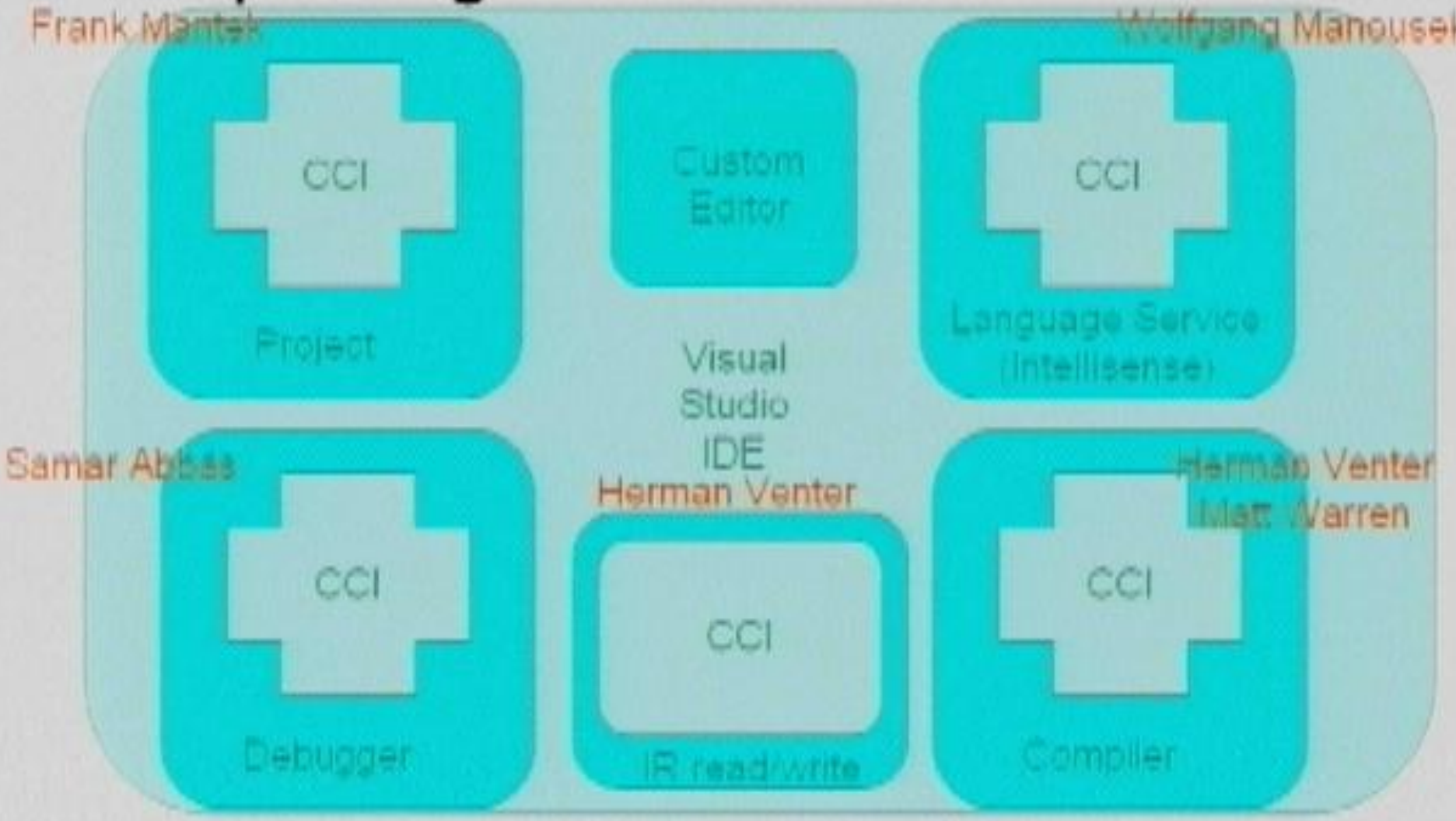


Not quite a semester project for undergrads

Better described as a collection of frameworks for writing VS packages



Better described as a collection of frameworks for writing VS packages



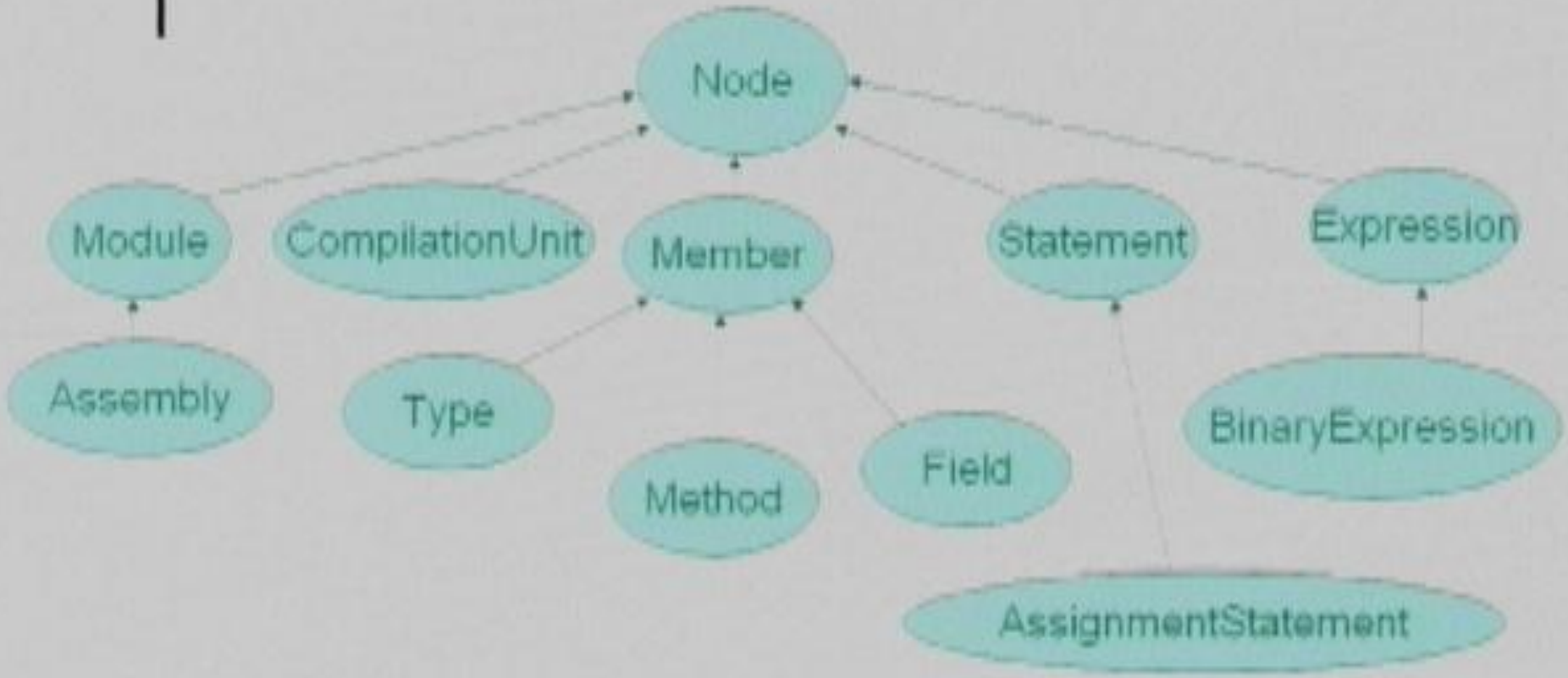
System.Compiler

- *The* class library for writing managed compilers and related tools on the CLI.
- Replaces
  - System.Reflection
  - System.Reflection.Emit
  - System.CodeDom
- Currently packaged as

  System.Compiler.dll

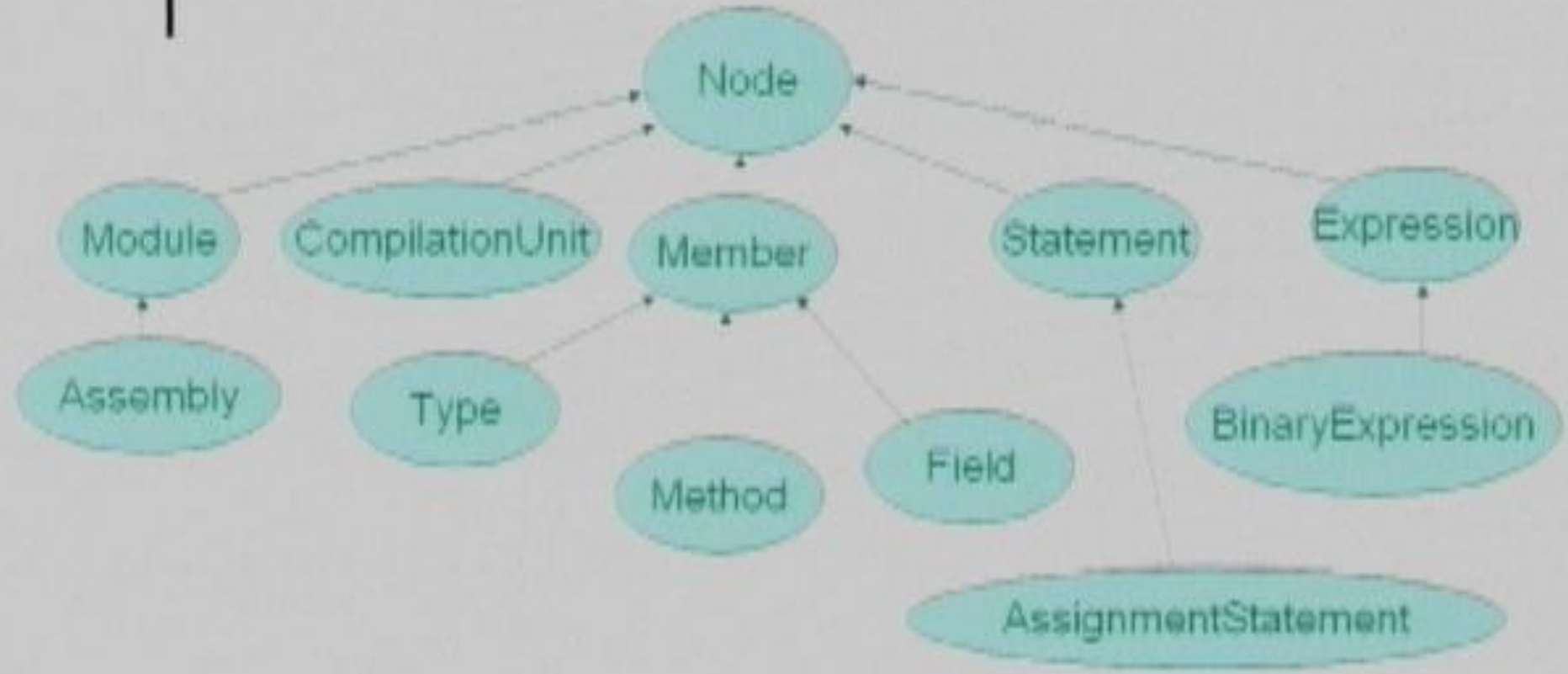
  System.Compiler.Framework.dll

Intermediate Representation



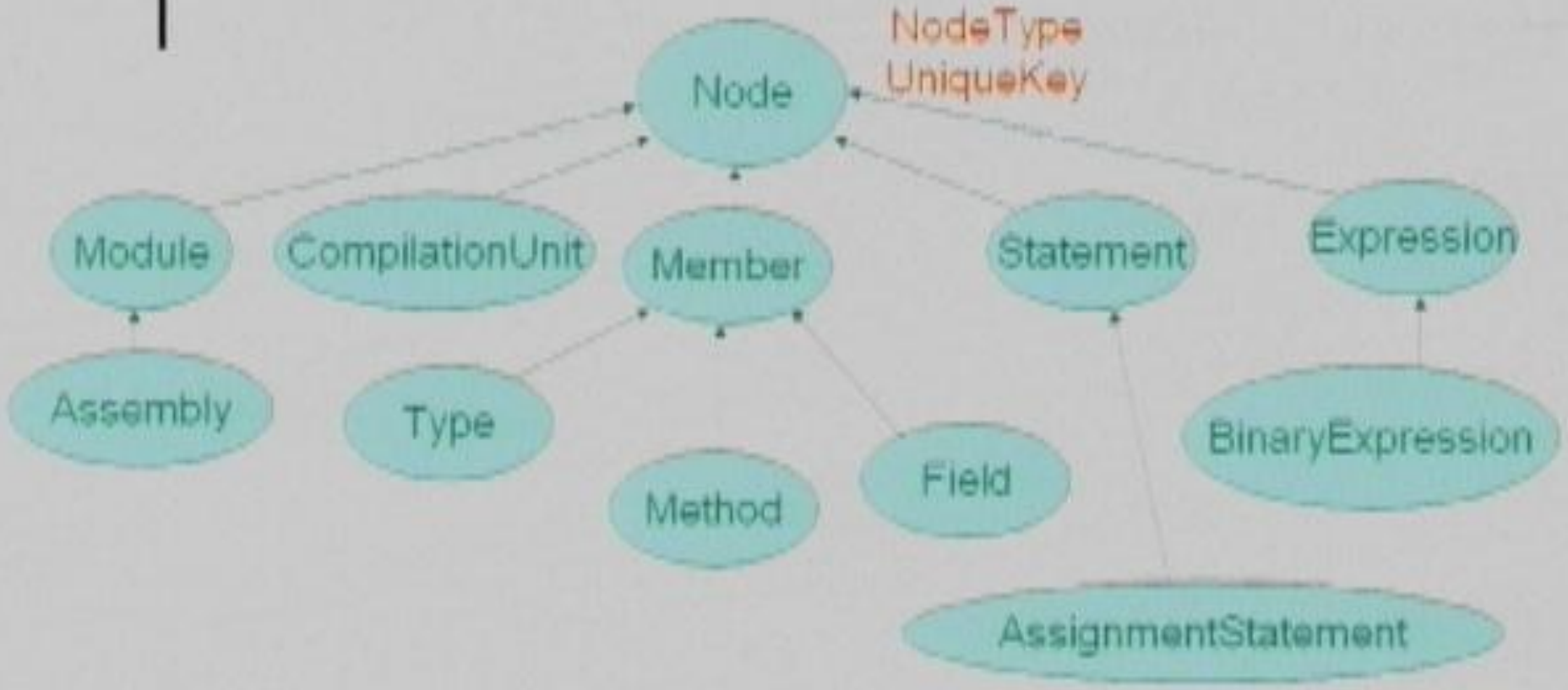
Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



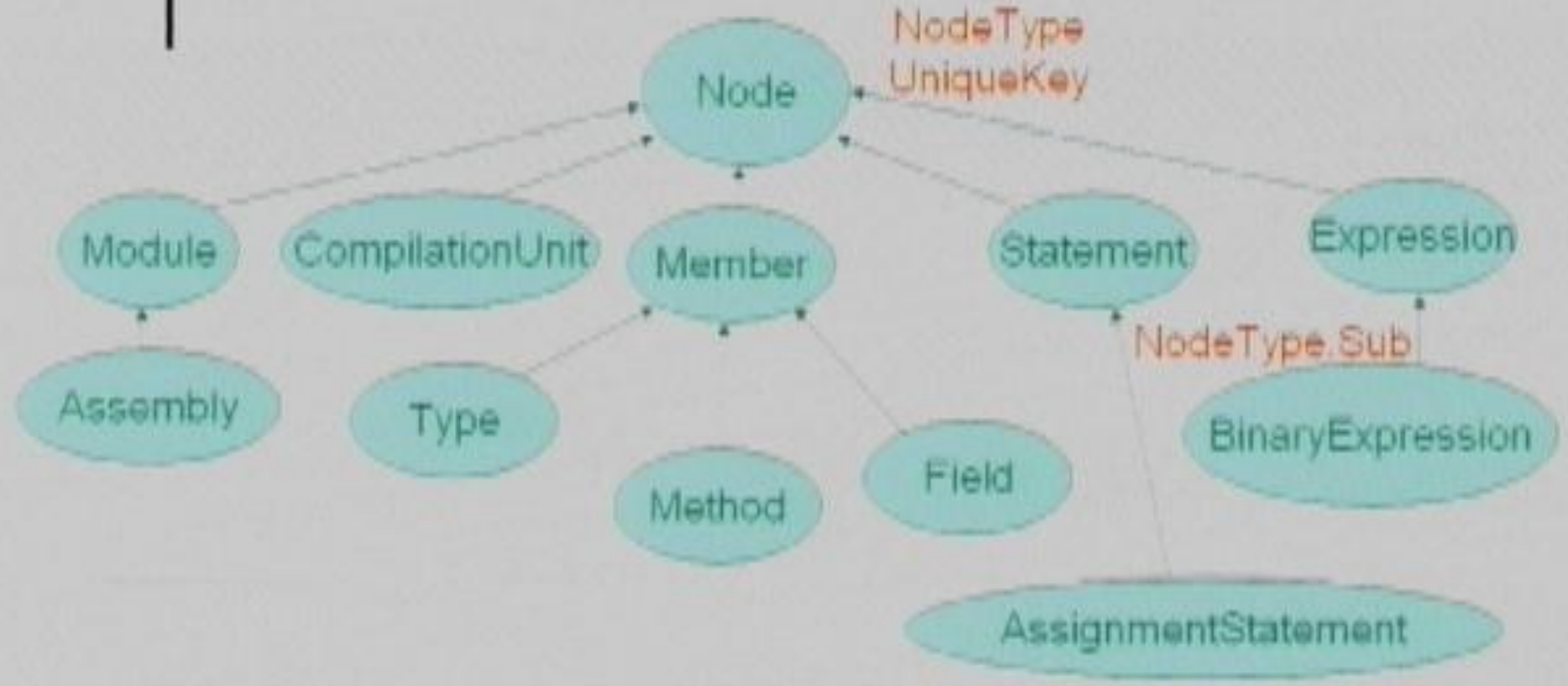
Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



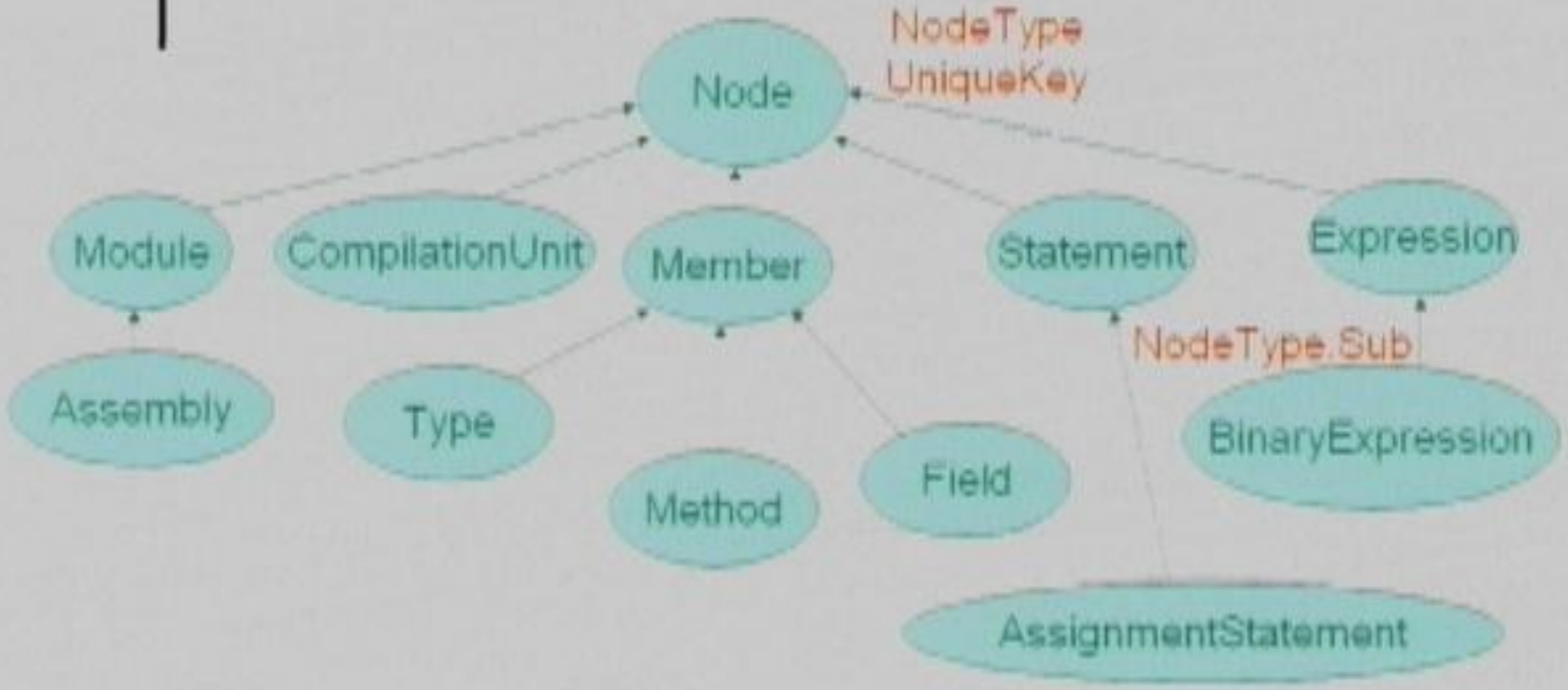
Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



Combines object model for metadata with object model for IL and traditional AST

Intermediate Representation



Abstracts over encoding issues (no ldc.i4.0, ldarg.0 and such like, just Literal, Parameter, Local, and so on)

Metadata reading and writing

- You can read in an assembly or module by calling: Assembly.GetAssembly(…) or Module.GetModule(…)
- You can write out an assembly or module by calling: Module.WriteModule(…)
- You can read or write memory streams, not just files
- PDB files also covered: see Node.SourceContext
- Writer requires IR to be "normalized"

Extending IR into your AST

- Simply extend Node, Statement, Expression, Type, etc. as appropriate.
- Make up your own NodeType enumeration with values that are distinct from System.Compiler.NodeType.
- All fields are public
- All methods are virtual

Visitors

```
/// Walks an IR, mutating it into a new form
public abstract class StandardVisitor: Visitor{
    public override Node Visit(Node node){
      if (node == null) return null;
      switch (node.NodeType){
        case NodeType.AddressDereference:
         return this.VisitAddressDereference(
                    (AddressDereference)node);
    …}
    public virtual VisitAddressDereference(…){
      if (addr == null) return null;
      addr.Address = this.VisitExpression(addr.Address);
      return addr;
    }
```

Visitors

```
/// Walks an IR, mutating it into a new form
public abstract class StandardVisitor: Visitor{
    public override Node Visit(Node node){
      if (node == null) return null;
      switch (node.NodeType){
        case NodeType.AddressDereference:
         return this.VisitAddressDereference(
                    (AddressDereference)node);
    …}
    public virtual VisitAddressDereference(…){
      if (addr == null) return null;
      addr.Address = this.VisitExpression(addr.Address);
      return addr;
    }
```

Extending Visitors to cope with your AST nodes

```
public abstract class StandardVisitor: Visitor{
    public override Node Visit(Node node){
      if (node == null) return null;
      switch (node.NodeType){…
        default:
          return this.VisitUnknownNodeType(node);
      }
    }
    public virtual Node VisitUnknownNodeType(Node node){
      Visitor visitor = this.GetVisitorFor(node);
      if (visitor == null) return node;
      return visitor.Visit(node);
    }
    public virtual Visitor GetVisitorFor(Node node){
      return (Visitor)node.GetVisitorFor(this,
      this.GetType().Name);
    }
```

Basic Visitors

- Visitor
- StandardVisitor
- StandardCheckingVisitor
- Duplicator
- Unstacker
- Specializer

Compilation

```
public abstract class Compiler :
  System.CodeDom.ICodeCompiler{
 public abstract void Compile(
  CompilationUnit compilationUnit,
  Class globalScope,
  ErrorNodeList errorNodes);
//ICodeCompiler method implementations
//Various helpers for command line
//parsing etc.
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
   Class globalScope, ErrorNodeList errorNodes){
   ErrorHandler errorHandler = new ErrorHandler(errorNodes);
   Looker looker = new Looker(globalScope, errorHandler);
   looker.VisitCompilationUnit(compilationUnit);
   Resolver resolver = new Resolver(errorHandler);
   resolver.VisitCompilationUnit(compilationUnit);
   Checker checker = new Checker(resolver, errorHandler);
   checker.VisitCompilationUnit(compilationUnit);
   Normalizer normalizer = new Normalizer();
   normalizer.VisitCompilationUnit(compilationUnit);
}
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Compilation

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Language Service

```
public abstract class LanguageService : IBabelService{
   public virtual void ColorLine(
      string line, IColorSink sink, ref int state)
   (…)
   public virtual IScope ParseSource(
      string text, IParseSink sink, ParseReason reason,
  int reserved)
   (…)
…)
```

Language Service

```
public abstract class LanguageService : IBabelService{…
  public Scanner scanner; //initialized by derived class
  public virtual void ColorLine(string line, IColorSink sink,
   ref int state){
    this.scanner.SetSource(line, 0);
    TokenInfo tokenInfo = new TokenInfo();
    while (scanner.ScanTokenAndProvideInfoAboutIt(tokenInfo,
   ref state)){
      sink.Colorize(tokenInfo.startIndex, tokenInfo.endIndex,
   (int)tokenInfo.color, (CharClass)tokenInfo.type,
   (int)tokenInfo.trigger);
    }
…}
```

Language Service

```
public abstract class LanguageService : IDabelService{…
  public virtual IScope ParseSource(
      string text, IParseSink sink, ParseReason reason, int reserved){
    switch(reason){
      …
      case ParseReason.ReasonCheck:
        this.sourceText = text;
        AuthoringSink asink = this.GetAuthoringSink(sink, reason);
        ErrorNodeList errors = new ErrorNodeList();
        this.ParseAndAnalyzeSource(text, errors, asink);
        this.ReportErrors(errors, asink);
        return this.GetAuthoringScope();
    }
    return null;
  }
…}
```

Language Service

```
public abstract class LanguageService : IBabelService{…
 public Parser parser; //initialized by derived class
 public virtual void ParseAndAnalyzeSource(string text, ErrorNodeList errors,
   AuthoringSink asink){
    Project project = this.GetProjectFor(asink);
    CompilationUnit compilationUnit =
    this.parser.ParseCompilationUnit(text, project.cOptions,
    errors, asink);
    ModuleReferenceList modRefs = compilationUnit.TargetModule.ModuleReferences;
    for (int i = 0, n = project.modules.Length; i < n; i++){
      Module m = project.modules[i];
      if (m.Name == asink.FileName) continue;
      modRefs.Add(new ModuleReference(m.Name, m));
    }
    this.AnalyzeParseTree(compilationUnit, this.GetGlobalScope(project), errors);
    compilationUnit.TargetModule.Name = asink.FileName;
    project.UpdateWith(compilationUnit.TargetModule);
 }
…
}
```

Language Service

```
public abstract class LanguageService : IBabelService{…
 public Parser parser; //initialized by derived class
 public virtual void ParseAndAnalyzeSource(string text, ErrorNodeList errors,
   AuthoringSink asink){
   Project project = this.GetProjectFor(asink);
   CompilationUnit compilationUnit =
   this.parser.ParseCompilationUnit(text, project.cOptions,
   errors, asink);
   ModuleReferenceList modRefs = compilationUnit.TargetModule.ModuleReferences;
   for (int i = 0, n = project.modules.Length; i < n; i++){
     Module m = project.modules[i];
     if (m.Name == asink.FileName) continue;
     modRefs.Add(new ModuleReference(m.Name, m));
   }
   this.AnalyzeParseTree(compilationUnit, this.GetGlobalScope(project), errors);
   compilationUnit.TargetModule.Name = asink.FileName;
   project.UpdateWith(compilationUnit.TargetModule);
 }
…
}
```

Language Service

```
public virtual void ParseAndAnalyzeSource(string text, ErrorNodeList
    errors, AuthoringSink asink){
    Project project = this.GetProjectFor(asink);
    CompilationUnit compilationUnit =
    this.parser.ParseCompilationUnit(text, project.cOptions, errors,
    asink);
    ModuleReferenceList modRefs =
    compilationUnit.TargetModule.ModuleReferences;
     for (int i = 0, n = project.modules.Length; i < n; i++){
       Module m = project.modules[i];
       if (m.Name == currentFileName) continue;
       modRefs.Add(new ModuleReference(m.Name, m));
     }
    this.AnalyzeParseTree(compilationUnit,
    this.GetGlobalScope(project), errors);
    compilationUnit.TargetModule.Name = asink.FileName;
    project.UpdateWith(compilationUnit.TargetModule);
}
```

Language Service

```
public virtual void ParseAndAnalyzeSource(string text, ErrorNodeList
    errors, AuthoringSink asink){
    Project project = this.GetProjectFor(asink);
    CompilationUnit compilationUnit =
    this.parser.ParseCompilationUnit(text, project.cOptions, errors,
    asink);
    ModuleReferenceList modRefs =
    compilationUnit.TargetModule.ModuleReferences;
    for (int i = 0, n = project.modules.Length; i < n; i++){
      Module m = project.modules[i];
      if (m.Name == currentFileName) continue;
      modRefs.Add(new ModuleReference(m.Name, m));
    }
    this.AnalyzeParseTree(compilationUnit,
    this.GetGlobalScope(project), errors);
    compilationUnit.TargetModule.Name = asink.FileName;
    project.UpdateWith(compilationUnit.TargetModule);
}
```

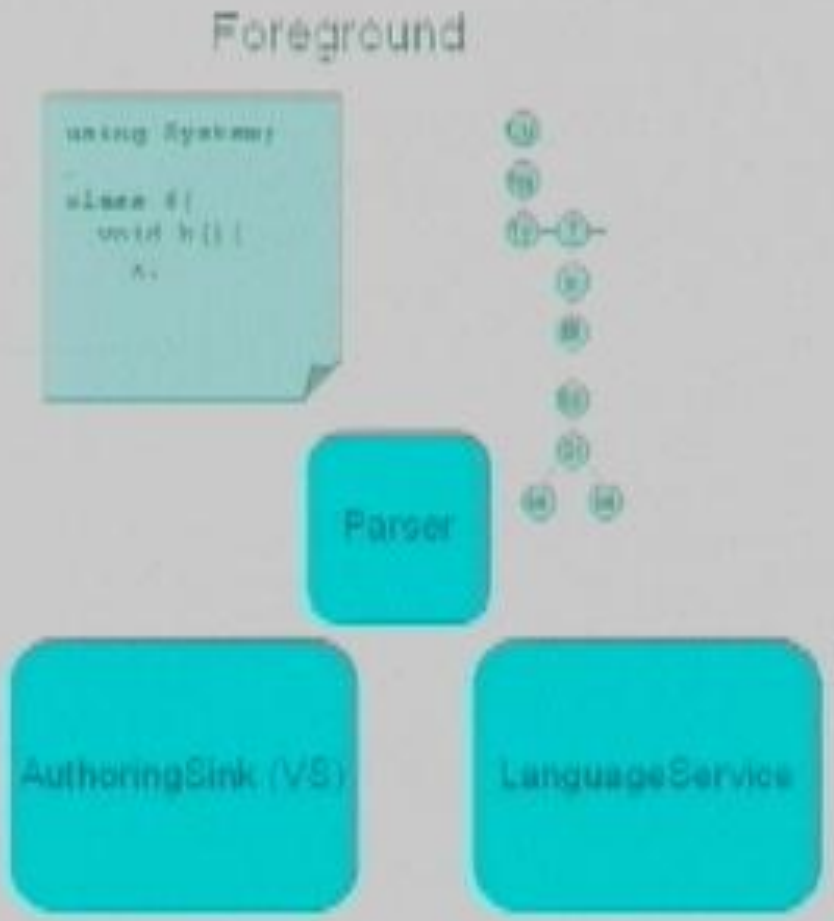
Language Service

```
public override void AnalyzeParseTree(CompilationUnit
  compilationUnit, Class globalScope, ErrorNodeList errors){
    ErrorHandler errorHandler = new ErrorHandler(errors);
    globalScope.DeclaringModule = compilationUnit.TargetModule;
    Looker looker = new Looker(globalScope, errorHandler);
    looker.VisitCompilationUnit(compilationUnit);
    Resolver resolver = new Resolver(errorHandler);
    resolver.VisitCompilationUnit(compilationUnit);
    Checker checker = new Checker(resolver, errorHandler);
    checker.VisitCompilationUnit(compilationUnit);
    //No Normalization. I.e. no code generation
}
```

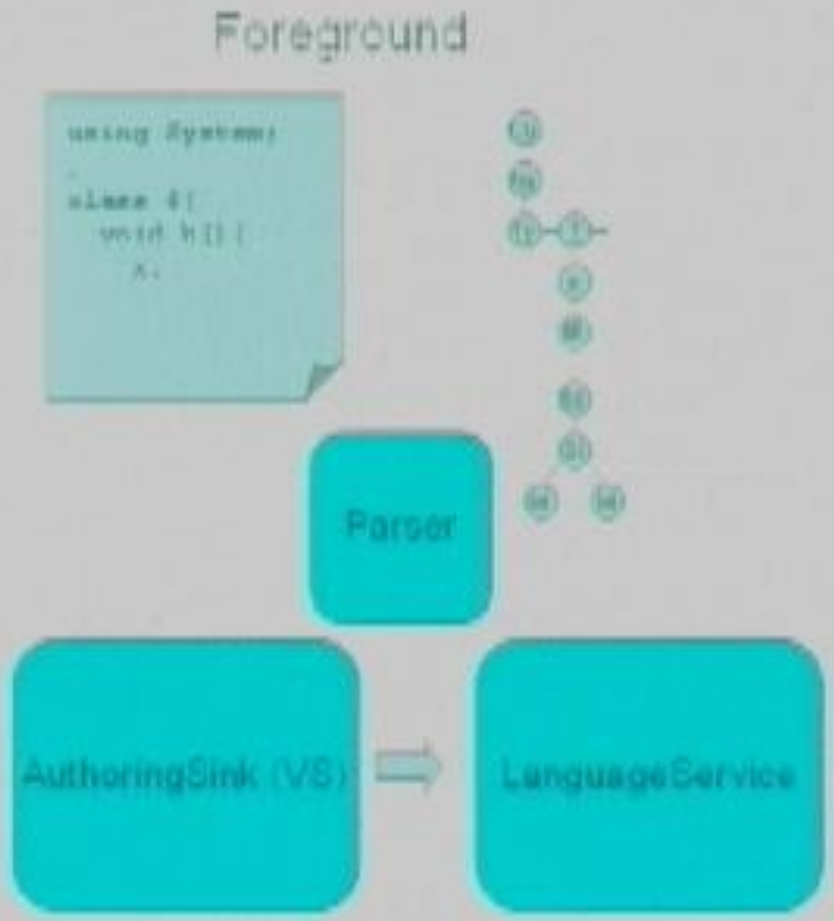
Language Service

```
public virtual IScope ParseSource(
     string text, IParseSink sink, ParseReason reason, int reserved){
  switch(reason){
    case ParseReason.ReasonMatchBraces:
    case ParseReason.ReasonHighlightBraces:
    case ParseReason.ReasonQuickInfo:
    case ParseReason.ReasonMemberSelect:
    case ParseReason.ReasonMethodTip:
    case ParseReason.ReasonAutos:{
      AuthoringSink asink = this.GetAuthoringSink(sink, reason);
      // In these cases we get only a part of the text
      this.partialCompilationUnit = this.parser.ParseCompilationUnit(text, new
   CompilerParameters(), new ErrorNodeList(), asink);
      break;
     }
     ...
  }
  return null;
}
```

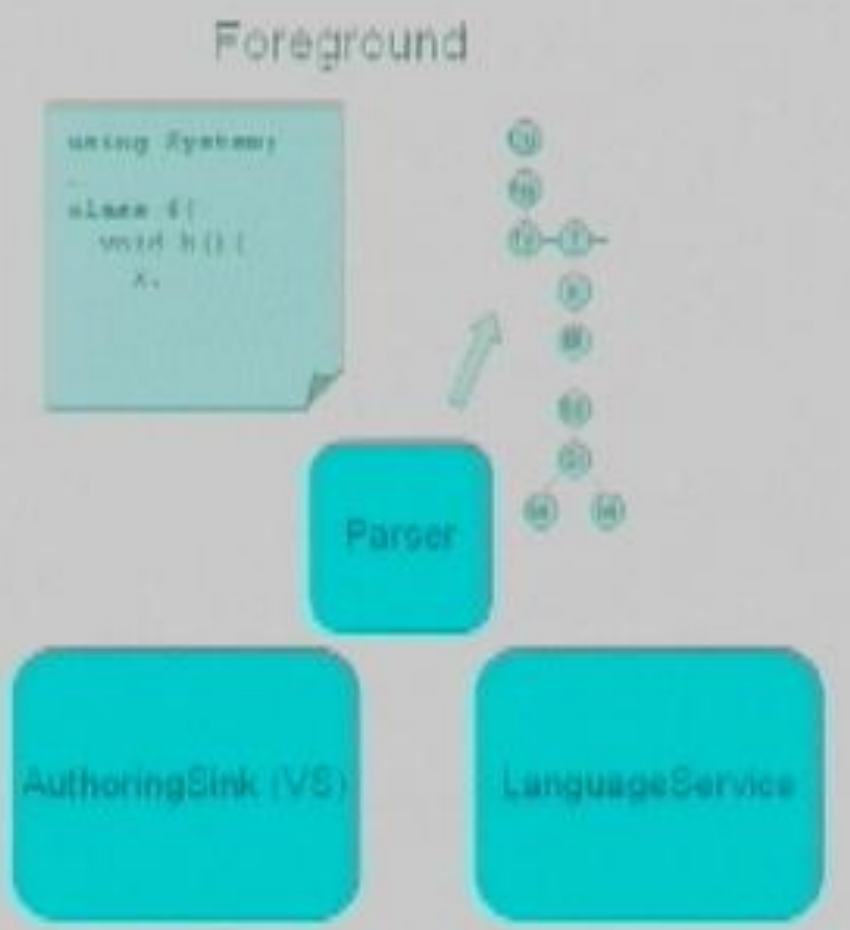
Language Service



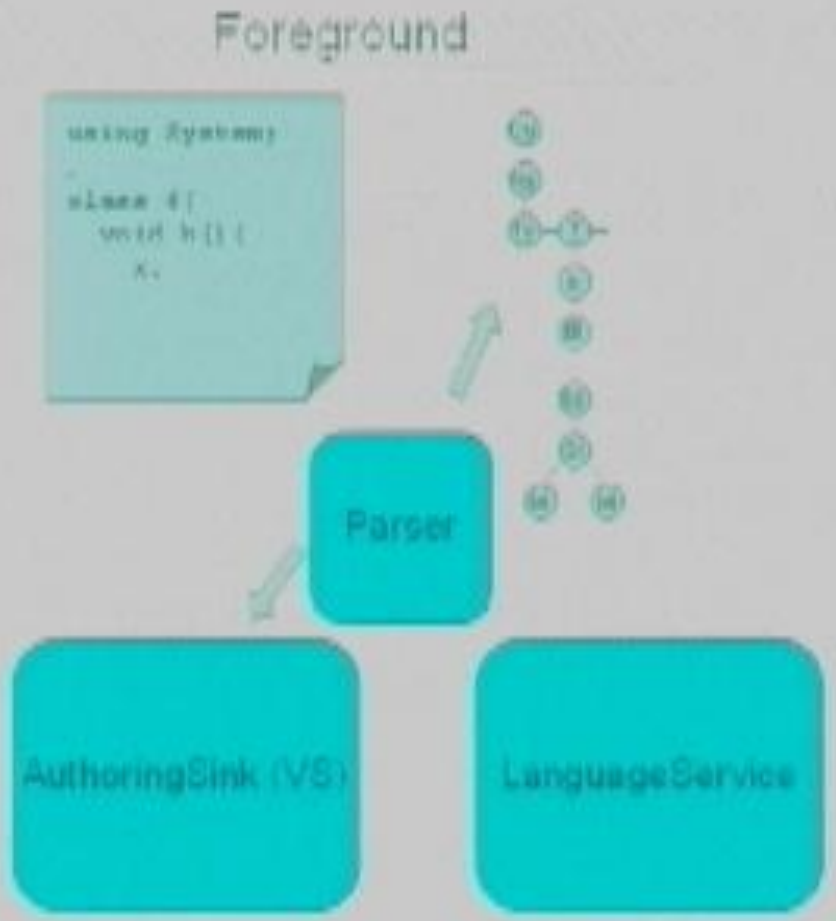
Language Service



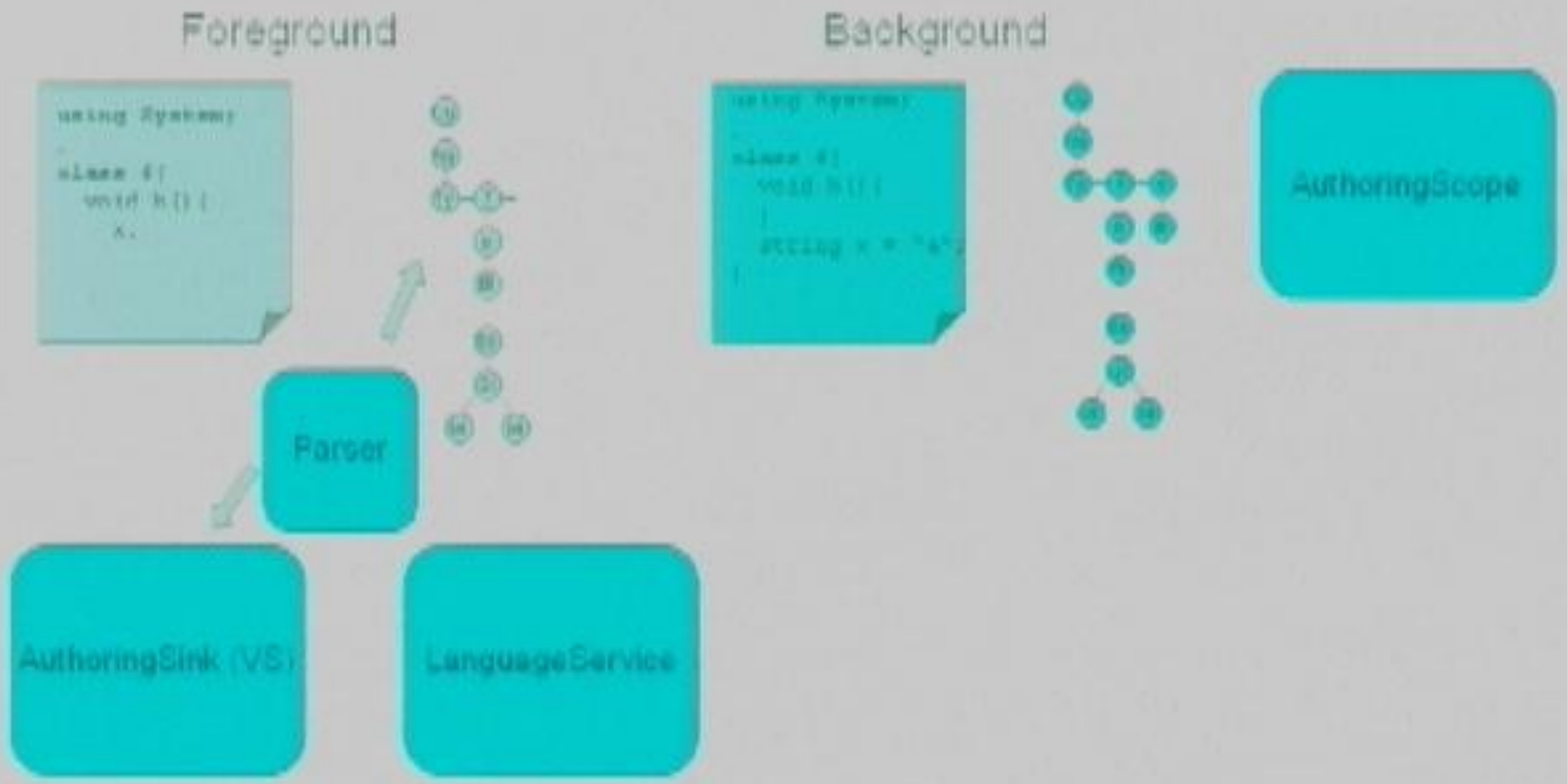
Language Service



Language Service



Language Service



Language Service

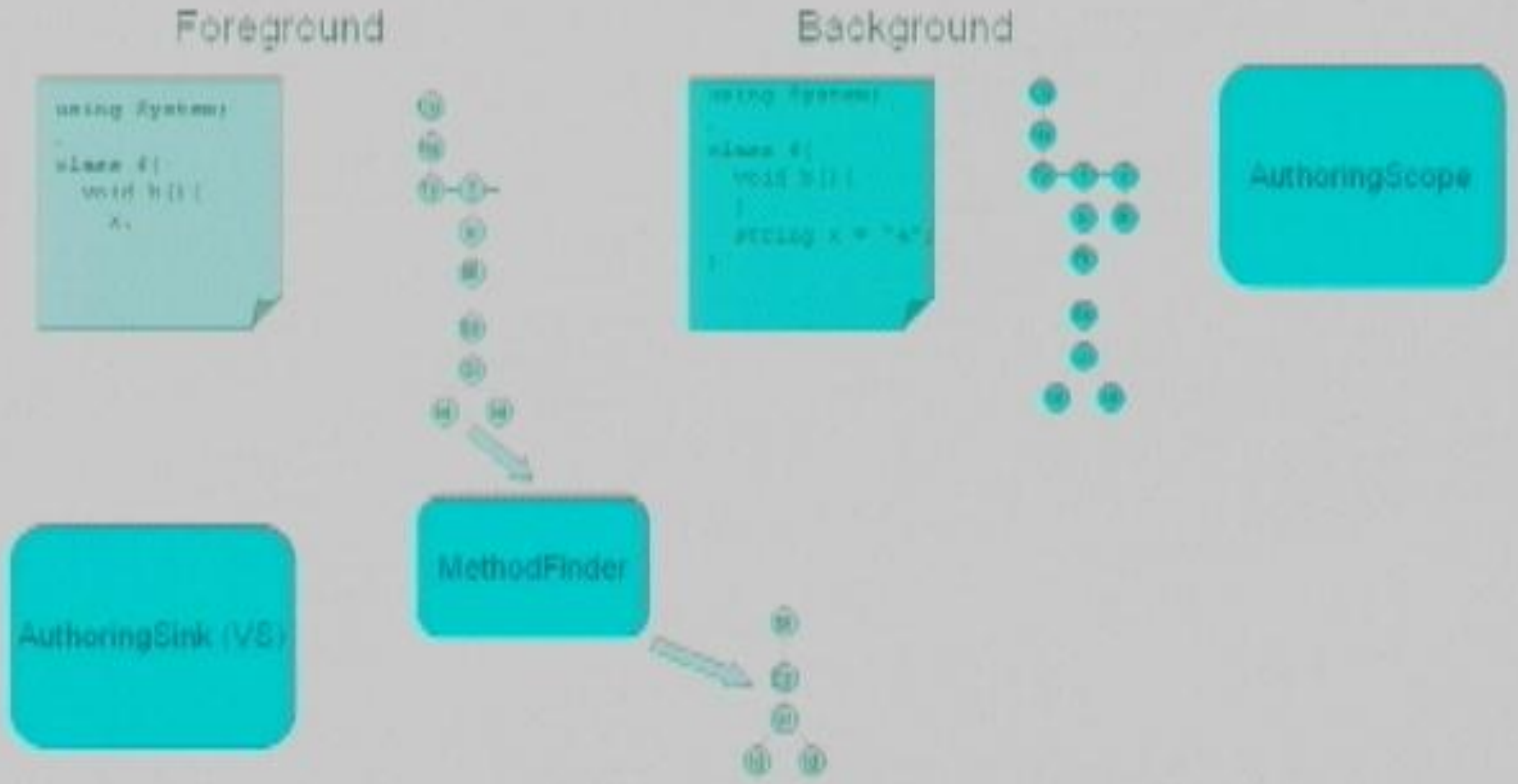
Foreground

Background

AuthoringScope

AuthoringSink (VS)

Language Service



Language Service

Foreground

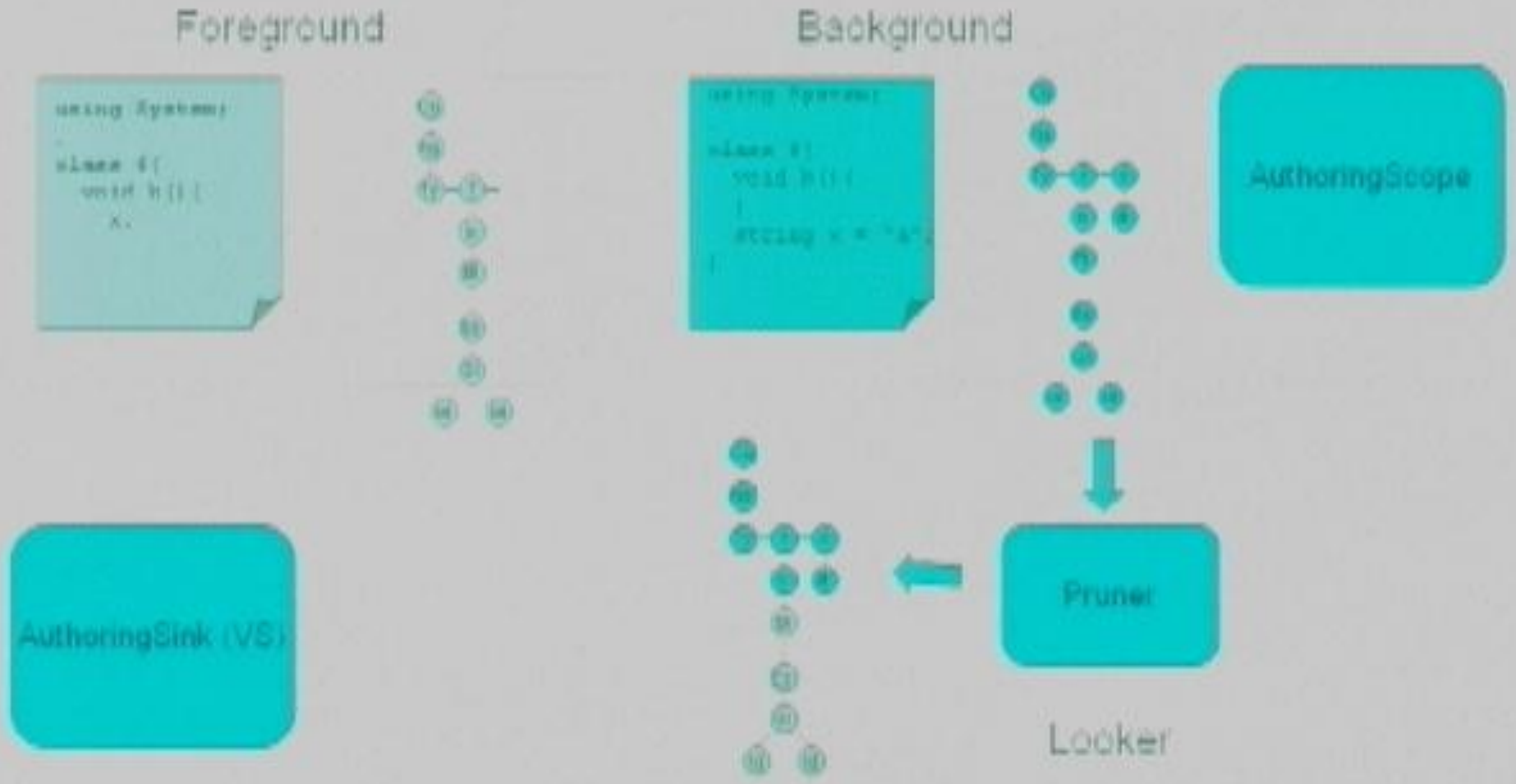
Background

AuthoringScope

Pruner

AuthoringSink (VS)

Language Service



Language Service

Foreground

Background

AuthoringScope

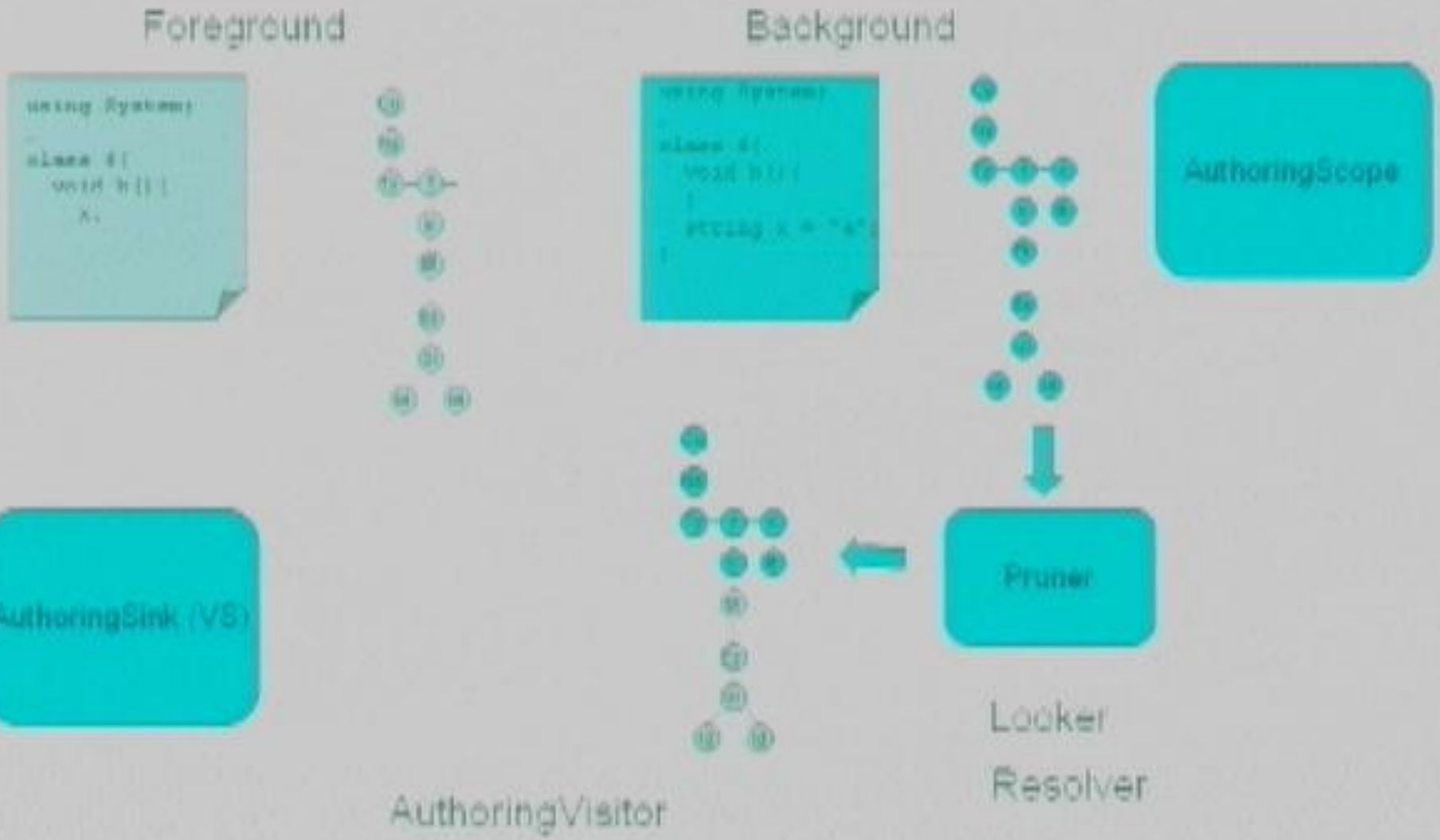
AuthoringSink (VS)

Pruner

Looker

Resolver

Language Service



Project

- Still in early stage of development
- Uses Compiler
- Provides information to Language Service
- Otherwise a huge chunk of code in a world of its own

Debugger

- Uses compiler to compile expressions
- Uses ErrorHandler to format output
- Provides an interpreter for IL, which is used by the compiler to do constant folding.
- This takes care of most language specific issues

Custom Checkers

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  this.RunCustomCheckers(compilationUnit, errorHandler);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Custom Checkers

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  this.RunCustomCheckers(compilationUnit, errorHandler);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Custom Checkers

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  this.RunCustomCheckers(compilationUnit, errorHandler);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Custom Checkers

```
public void RunCustomCheckers(
  CompilationUnit compilationUnit,
  ErrorHandler errorHandler)
{
  foreach (StandardCheckingVisitor cChecker
    in this.GetCustomCheckers(errorHandler))
    cChecker.VisitCompilationUnit(compilationUnit);
}
```

Post Build Checkers

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
  this.RunPostBuildCheckers(compilationUnit.targetModule,
   errorHandler);
}
```

Embedded Foreign Languages

- All CCI languages use the same base types for Type, Expression, Statement, Block, etc.
- This means that IR trees produced by different parsers can be grafted together into a single tree.
- The single tree is compiled co-operatively by separate visitor instances.
- This requires some help from VisitUnknownNodeType.

Embedded foreign languages

```
public virtual Node VisitUnknownNodeType(Node node){
  public virtual Node VisitUnknownNodeType(Node node){
    Visitor visitor = this.GetVisitorFor(node);
    if (visitor == null) return node;
    if (this.callingVisitor != null)
      this.callingVisitor.TransferStateTo(visitor);
    this.TransferStateTo(visitor);
    node = visitor.Visit(node);
    visitor.TransferStateTo(this);
    if (this.callingVisitor != null)
      visitor.TransferStateTo(this.callingVisitor);
    return node;
  }
}
public virtual Visitor GetVisitorFor(Node node){
  return (Visitor)node.GetVisitorFor(this,
   this.GetType().Name);
}
```

Compiler Plugins

```
public override void Compile(CompilationUnit compilationUnit,
    Class globalScope, ErrorNodeList errorNodes){
  ErrorHandler errorHandler = new ErrorHandler(errorNodes);
  Looker looker = new Looker(globalScope, errorHandler);
  looker.VisitCompilationUnit(compilationUnit);
  Resolver resolver = new Resolver(errorHandler);
  resolver.VisitCompilationUnit(compilationUnit);
  Partitioner partitioner = new Partitioner();
  partitioner.VisitCompilationUnit(compilationUnit);
  Checker checker = new Checker(resolver, errorHandler);
  checker.VisitCompilationUnit(compilationUnit);
  Normalizer normalizer = new Normalizer();
  normalizer.VisitCompilationUnit(compilationUnit);
}
```

Links

http://xsharp

mailto:hermanv@microsoft.com

http://pgm/cci

http://research.microsoft.com/programs/europe/events/dotnetcc/Version2/Crash%20Course.ppt

SDPORT=wdinc:9100, root /xsharp